CR 90.016

NCEL

Contract Report

July 1990

An Investigation Conducted by:
Joseph A. Landers
Department of Civil Engineering
University of California, Berkeley

A SOFTWARE DEVELOPMENT SPECIFICATION FOR NONLINEAR STRUCTURAL ANALYSIS

ABSTRACT This report describes a programming environment for structural engineering computations. Offering many advantages over the current state of technology in this area, the software system discussed here is highly flexible and portable. In addition to carrying out sophisticated calculations efficiently on today's engineering workstations, the environment can also exploit the power of larger computers by linking tasks over a local area network. Furthermore, the system is programmable and extensible. Finally, the software system may be integrated with existing programs such as finite element codes and mathematical libraries.

Keywords: Software engineering, interpreters, (KR)

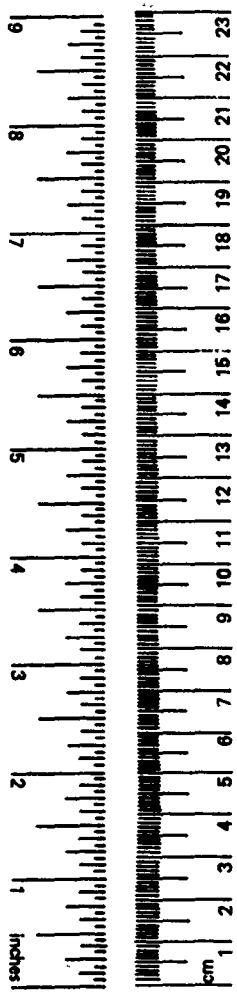
NAVAL CIVIL ENGINEERING LABORATORY PORT HUENEME CALIFORNIA 93043

METRIC CONVERSION FACTORS

Approximate Conversions to Metric Measures

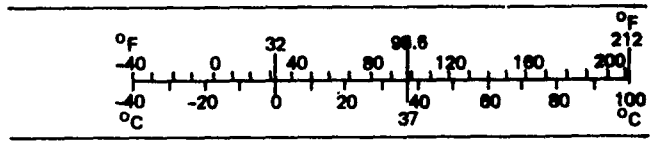
Symbol	When You Know	Multiply by	To Find	Symbol
		LENGTH		
in	inches	*2.5	centimeters	cm
ft	feet	30	centimeters	cm
yd	yards	0.9	meters	m
mi	miles	1.6	kilometers	km
		AREA		
in^2	square inches	6.5	square centimeters	cm^2
ft^2	square feet	0.09	square meters	m^2
yd^2	square yards	0.8	square meters	m^2
mi^2	square miles	2.6	square kilometers	km^2
	acres	0.4	hectares	ha
		MASS (weight)		
oz	ounces	28	grams	g
lb	pounds	0.45	kilograms	kg
	short tons (2,000 lb)	0.9	tonnes	t
		VOLUME		
tsp	teaspoons	5	milliliters	ml
Tbsp	tablespoons	15	milliliters	ml
fl oz	fluid ounces	30	milliliters	ml
c	cups	0.24	liters	l
pt	pints	0.47	liters	l
qt	quarts	0.95	liters	l
gal	gallons	3.8	liters	l
ft^3	cubic feet	0.03	cubic meters	m^3
yd^3	cubic yards	0.76	cubic meters	m^3
		TEMPERATURE (exact)		
°F	Fahrenheit temperature	5/9 (after subtracting 32)	Celsius temperature	°C

*1 in = 2.54 (exactly). For other exact conversions and more detailed tables, see NBS Misc. Publ. 286, Units of Weights and Measures, Price $2.25, SD Catalog No. C13.10:286.



Approximate Conversions from Metric Measures

Symbol	When You Know	Multiply by	To Find	Symbol
		LENGTH		
mm	millimeters	0.04	inches	in
cm	centimeters	0.4	inches	in
m	meters	3.3	feet	ft
m	meters	1.1	yards	yd
km	kilometers	0.6	miles	mi
		AREA		
cm^2	square centimeters	0.16	square inches	in^2
m^2	square meters	1.2	square yards	yd^2
km^2	square kilometers	0.4	square miles	mi^2
ha	hectares (10,000 m^2)	2.5	acres	
		MASS (weight)		
g	grams	0.035	ounces	oz
kg	kilograms	2.2	pounds	lb
t	tonnes (1,000 kg)	1.1	short tons	
		VOLUME		
ml	milliliters	0.03	fluid ounces	fl oz
l	liters	2.1	pints	pt
l	liters	1.06	quarts	qt
l	liters	0.26	gallons	gal
m^3	cubic meters	35	cubic feet	ft^3
m^3	cubic meters	1.3	cubic yards	yd^3
		TEMPERATURE (exact)		
°C	Celsius temperature	9/5 (then add 32)	Fahrenheit temperature	°F



REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-018

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1990	Final; September 1987 thru October 1988

4. TITLE AND SUBTITLE

A SOFTWARE DEVELOPMENT SPECIFICATION FOR NONLINEAR STRUCTURAL ANALYSIS

5. FUNDING NUMBERS

PE - YR023.03.01.005
PR - RM33F60-A2-06-010

6. AUTHOR(S)

Joseph A. Landers

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESSE(S)

Department of Civil Engineering
Division of Structural Engineering, Mechanics & Materials
University of California, Berkeley
Berkeley, CA 94720

8. PERFORMING ORGANIZATION REPORT NUMBER

CR-90.016

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESSE(S)

Office of Naval Technology
800 N. Quincy Street
Arlington, VA 22217-5000

/ Naval Civil Engineering Laboratory
Port Hueneme, CA 93043-5003

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report describes a programming environment for structural engineering computations. Offering many advantages over the current state of technology in this area, the software system discussed here is highly flexible and portable. In addition to carrying out sophisticated calculations efficiently on today's engineering workstations, the environment can also exploit the power of larger computers by linking tasks over a local area network. Furthermore, the system is programmable and extensible. Finally, the software system may be integrated with existing programs such as finite element codes and mathematical libraries.

14. SUBJECT TERMS

Structural engineering, finite element analysis, software engineering, software development, programming environment, interpreter, virtual machine

15 NUMBER OF PAGES

18

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

TABLE OF CONTENTS

A Software Development Specification for Nonlinear Structural Analysis

Joseph A. Landers*

Abstract

This report describes a programming environment for structural engineering computations. Offering many advantages over the current state of technology in this area, the software system discussed here is highly flexible and portable. In addition to carrying out sophisticated calculations efficiently on today's engineering workstations, the environment can also exploit the power of larger computers by linking tasks over a local area network. Furthermore, the system is programmable and extensible. Finally, the software system may be integrated with existing programs such as finite element codes and mathematical libraries.

1 INTRODUCTION

For many years computers have found applications in the solution of structural engineering problems. The machines were, for the most part, utilized during the analysis phase of a project. Often, the process consisted of preparing an input file with a keypunch or generating information with a text editor. After presenting this data to a large and complex structural analysis program in a batch environment, a substantial volume of results would be returned to the engineer. While making the most efficient use of limited computer resources, this method of interaction was often tedious and prone to expensive errors. For example, one simple mistake due to a typing error might have significantly changed the meaning of the input which could have negated hours of computation. The batch oriented environment also made the development and testing of new methodologies and algorithms very difficult. Even when presented with an interactive text editor, the construction and debugging of new concepts and ideas often involved many iterations of the "edit, compile and debug" cycle.

Over the past several years, significant changes in the computer market have created new computational opportunities for structural engineers. The availability of low cost workstations equipped with powerful 32-bit processors, high resolution graphic displays and inexpensive networking facilities can be expected to make a significant impact on the way in which engineering problems are formulated and solved. While the cost of these computer systems has been dropping rapidly, the amount of raw processing power has been growing almost geometrically. This has been especially true in the highly competitive market of general purpose computer systems. Even the large scale vector and parallel processing machines have now become more accessible to general engineering applications. Significantly, both hardware and software vendors are beginning to come to a consensus on standards for the tools they provide. For example, there are computer networking standards for distributed file systems such as NFS [27], standards for inter-machine communication such as RPC [6], and standards for graphic display and interaction such as PHIGS [5], GKS [12] and the X Window System [28]. There are even standards under develop-

*Department of Civil Engineering, Division of Structural Engineering, Mechanics and Materials, University of California, Berkeley.

ment for operating systems such as POSIX [11]. From a buyer's standpoint, this agreement among vendors has helped to stabilize the computer workstation market and make the choices of a particular brand of hardware and software less important. Now, not only does the buyer have a better basis for comparing different systems, but the decision to purchase one vendor's equipment over another's no longer entails a lifelong commitment.

Unfortunately, the advances in structural engineering software have not kept pace with the rapid changes in the computer marketplace. Certainly, some applications which were developed and used on computer systems popular in the past have been modified to run on today's workstations [19]. Some software, developed in the commercial sector, has been radically modified and enhanced to exploit the capabilities of the new hardware [7]. A few new packages have even been written [25]. Yet, while these systems provide functionality in the form of large and monolithic programs, little innovative software is available to the research and development community of structural engineers. Because this group relies upon advances in several diverse areas such as mathematics, numerical analysis and computer science as well as engineering, they require a computational environment that is responsive to changes in the state of the art. Since new techniques and algorithms must be tested and debugged as they are implemented, these researchers also require a system that provides a high degree of flexibility and interaction.

This report provides a somewhat detailed description of a software environment for structural engineering applications. While the system discussed here is primarily intended for the design and implementation of new methodologies and techniques in finite element analysis, this software can also be extended and applied to commercial production situations as well. The original intent of this report was to furnish a description of a theoretical system, with the implementation coming at some later time. However, during the course of preparing this account, it became clear that at least a portion of the programming environment would prove of great value in the author's current research in the development of a new finite element model for shell structures. Hence, while the software system described here has not been completely constructed, some major sections of the environment have already been implemented at the University of California, Berkeley. Several of the other ideas described here have, for the most part, already been included in other systems outside of the discipline of structural engineering [9, 13,29]. They should not, therefore, represent a great deal of effort to add to the already existing collection of computer code. Hence, the detailed account of the software system described here is more than just a theoretical exercise, it contains the recommendations based upon partially implementing the environment outlined in [14].

2 HISTORICAL PERSPECTIVE

Although this software system contains many unique and innovative ideas in the context of structural engineering applications, these concepts are related to a large body of previous work. The programming environment described here not only builds upon the work done by others in the area of structural engineering, but it also draws upon the experience of those from outside of this field, including such diverse areas of mathematics and computer science. Many valuable insights can be gained by studying how previous researchers have investigated and solved problems related to this topic.

This programming environment owes much to three earlier structural engineering software systems. The first, the Integrated Civil Engineering System or ICES [17], was a very ambitious collection of programs developed during the 1960's at the Massachusetts Institute of Technology. ICES sought to provide a common set of utilities not only for structural computations, but also for other areas of civil engineering as well. Considering the capabilities of the computer hardware available at that time, the system was quite sophisticated. Applications used a crude keyword-oriented command parser, and were able to extract and manipulate objects located in simple tables. Later, a commercial vendor extended the functionality of the software to include automated design facilities [22]. This vendor still supports a derivative of the work today. The second program which had a major influence on the current software system was the Problem Oriented Language Organizer or POLO [18], developed during the 1970's at the University of Illinois. A POLO system for the analysis of engineering structures featured a high level of interaction in the manipulation of finite element models. The language was quite flexible and friendly. The third and final system to have influence on the design of the current environment was developed during the 1980's by the architectural firm of Skidmore, Owings and Merrill in Chicago, Illinois. While private firms have always developed software to solve problems particular to their business, this system had many interesting attributes. The Structural Data Management System or SDMS [16], was a well developed environment for the design and analysis of tall buildings. SDMS featured a high level of interaction in the form of both a fairly sophisticated command parser and graphic interface. The system also had the ability to share information with other application packages in-

cluding those oriented toward drafting and mechanical design.

Large, general purpose finite element codes have also had a significant influence on the overall design of the current programming environment. The system detailed in this report is not intended to replace these codes, but rather it seeks to augment them by integrating their operation with new capabilities. From the standpoint of the construction of a software system for structural engineering computations, these general purpose finite element codes are more interesting, not in terms of the activities they support but rather for their information requirements. Many popular general purpose computer codes are available for finite element analysis applications. Some of these include NASTRAN [19], ANSYS [7] and SAP [36]. These codes provide a library of finite element models and have a variety of analytical capabilities including both static and dynamic response options and perhaps a capacity for investigating non-linear behavior.

A few research systems for mathematical and engineering calculations provide some other interesting perspectives. The CAL 78 [35] system is a matrix interpreter specifically oriented towards solving structural engineering problems. The MATLAB [23] system provides some additional sophisticated functionality in a more general mathematical setting. A large general purpose symbolic manipulation system, MACSYMA [21], has the ability to solve problems in differential and integral calculus, along with many other capabilities. The research code FEAP [33] is a hybrid between a general purpose finite element program and a structural engineering development system.

With the availability of engineering workstations, several projects developed by the computer science community also deserve some mention. These include the highly sophisticated Smalltalk [9] system, which is a complete programming environment for software development and a programming support system for the Ada [1] language which has been marketed by several vendors.

Ideally, the system described in this report should be an aggregate of all of the best features of the programs reviewed here. However, given the constraints on both the time and resources available, this new programming environment for structural engineering computations can only approximate this ambitious goal. The new system is still very powerful and flexible. It combines the response of an interactive program with the speed and efficiency of a batch-oriented system. It is programmable and extensible. The intent of this report is to provide a common framework where all of the separate ideas and concepts presented in this section can be combined for the benefit of structural engineers. It is this integration that is especially important for engineers who work at the leading edge of computational technology.

3 TERMINOLOGY

In this section several technical terms and concepts which are frequently used throughout this report will be defined and discussed. The intent of this section is to provide a common reference point to these ideas so that the reader can have a better understanding of the underlying structure and philosophy of the current programming environment. Since it is assumed the reader is already familiar with the various technical terminology in the realm of structural engineering, only a few new concepts from field of computer science will be presented. Note that while the definitions presented here are quite general, they may or may not represent the meanings in a more general computer science context.

3.1 Interpreter

An interpreter provides a mechanism for translating input provided by the user into actions which are usually executed immediately by a machine. A compiler, in contrast to an interpreter, generally stores this translation in a file, possibly in some different internal form, for execution by a machine at a later time.

Interpreters allow new algorithms to be written interactively. They can provide immediate feedback on the implementation. While an interpreter may not always provide the most efficient means for the execution of a program, they usually can offer very high levels of debugging support. The execution of code resulting from processing a user's input is most often done by another piece of software called a virtual machine.

Input translation into a form suitable for execution on a machine is done in five phases. During the first two phases, the input is scanned and parsed. Here, the information is broken into pieces called tokens. These tokens are recognized on a syntactic level during the second phase in order to determine which programming structure is being represented. For example, consider the following line of code.

```
sum = a + 2.0
```

The tokens are "**sum**," "**=**," "**a**," "**+**," and "**2.0**." Syntactically, this line represents the form of a binary addition followed by an assignment. While the second phase analyzes the form of the input, the next translation phase studies its meaning. This third phase, se-

mantic analysis, may provide extra information on how a task is to ultimately be carried out on the machine. Alternatively, semantic analysis may also find errors in the application of the programming constructs based upon the context of their use. For example, the following line of code is syntactically correct as an assignment of the sum of two constants, but it is semantically in error because character strings may not be added to integer values.

```
a = 1 + "abcdefg"
```

The fourth phase performs optimizations on the code. This is critical in a programming environment for structural engineering calculations because such tasks are generally very computationally intensive. For more details on this phase see [15]. The fifth and final phase consists of mapping the translation into a form suitable for execution on the machine. Here, for example, loops are converted to machine idioms for "test and branch." More details on all of these phases can be found in [2] or [37].

3.2 Language Design

Language design and implementation are important issues in any interpretive environment but they are particularly critical for the system described here. There are several reasons. First, the language interpreter serves as the primary mechanism of communication between the user and the computer. Hence, the environment must provide a flexible means of translating concise engineering descriptions into efficient actions. Second, the form of the language dictates how other portions of the system fit together. For this reason, the language must provide sufficient power to express a wide variety of concepts and ideas. Finally, in order to keep the environment accessible to a large pool of engineers who may work with the system only on occasion, the entire collection of software must be logically designed and implemented. If possible, it should be biased toward exploiting the user's existing knowledge of computer programming. For example, an environment would be much easier to learn and apply if the programming constructs and techniques were tied to an existing language such as **Fortran** rather than a language dialect unfamiliar to engineers.

Two important language concepts which hold very prominent places in the design of this environment for structural engineering computations have to do with the *type* and *scope* of variables. A *type* is categorized by a set of allowable values, a mechanism for specifying those values and a collection of permissible operations which use those values. The *scope* of a variable deals with the portion of program text where a given name has the same interpretation.

The current **Fortran** standard [3] allows only for a limited number of primitive data types. For the most part, these types are closely related to the underlying computer hardware and there is no mechanism for defining new data types. By contrast, the system described here not only supports **Fortran**'s simple types but it also allows the programmer to construct new data types by aggregating these basic types with any already existing data types. Essentially, this environment implements some of the features described in the proposed **Fortran 8x** language [4]. The ability to define and manipulate variables composed of aggregates of the basic types not only makes writing and debugging algorithms much more straightforward, but it also facilitates the integration of different applications under a common environment. For example, the programming environment may not only link to an existing finite element program coded in **Fortran**, but it may also communicate with a symbolic manipulation system written in the **LISP** language [34].

Under the existing standard, the scope of a variable in a **Fortran** program is limited to the function or subroutine where it is defined. Names of **common** blocks and program units are globally persistent. Furthermore, there is no facility for hiding data definitions. While program units may associate storage through **common** statements, there is no standard mechanism for maintaining variable names across program units. By contrast, the programming environment described here allows not only data hiding, but also contains provisions for named global storage.

3.3 Virtual Machine

A virtual machine may be roughly defined as a complete computer system, including both the underlying hardware and its software, implemented entirely by a computer program. Virtual machines mimic the hardware facilities supplied by a processor, memory and input/output actions as well as the computer's operating environment. They may also support other tasks such as local disk storage and access to network functions.

Virtual machines offer several advantages. First, they can provide a portable base for software development. Only the virtual machine itself must be ported to a new computer architecture or operating system. Existing applications which utilize the machine do not have to be modified. Second, these machines present a common interface to the application software. Hence, there tends to be greater uniformity for both the programmer and user. Third, virtual machines can be

quite flexible and often offer an alternative to large, monolithic programs. A user may pick and choose among options dynamically. There have been many successful environments based upon virtual machines including those described in [16], [31] and even [24].

3.4 Computer Networks

A few years ago computer networks usually consisted of small groups of machines sharing relatively tiny pieces of information around a single office or building. Alternatively, they also described a collection of computers shipping data over a leased telephone line. While these networks provided a valuable service in transferring blocks of information to remote sites, they were somewhat expensive to operate reliably and they required extensive user intervention.

Today, computer networks have much more powerful capabilities. Well defined standards now exist [32] so that many very different computers can share not only individual data but also physical resources over a wide geographical area. For example, not only can objects such as simple collections of files be transparently and instantaneously accessed, but entire databases may also be made available.

Furthermore, as an outgrowth of the ability to share physical resources over a computer network there are also some sophisticated communication facilities now available. One of these, the public domain Remote Procedure Call or RPC mechanism [6], allows user level software to dynamically call procedures on another machine. This powerful operation is supported by the system described in this report.

3.5 Dynamic Binding

Dynamic binding is a relatively new technique of combining pieces of software together in a manner which allows a great deal of flexibility on today's general purpose virtual memory computers. In standard practice, a programmer often describes the actions a section of software should take by providing a description in a high level language such as **Fortran** or **C**. This description is then compiled into a format suitable for the underlying computer hardware. At a later time, different modules are linked or bound together resulting in a single monolithic executable image. This binding, which is only done once and lasts essentially forever, is known as static binding.

In today's virtual memory computer architectures, static binding can lead to gross inefficiencies in the execution of a program image. This is because these computer systems often bring in tiny pieces of a program to the machine's memory in segments called pages. These pages represent both the instructions and data of the executable image. In large codes, such as those commonly employed in finite element studies, there are often many portions of the program which are not used in a given analysis. Unfortunately, however, these unused areas must be transferred in and out of computer memory before the proper segments of the code are resident and available to the central processor. This unnecessary and often time consuming conduct has two detrimental effects on today's modern computer architectures. First, it causes the program to seize large amounts of valuable physical memory resources. This can negatively impact the behavior of other jobs running on the computer. Second, many computer systems dynamically move jobs out of physical memory and to secondary disk storage when central memory facilities become unavailable. This process is called swapping. Since these large programs capture many resources, the central processor unnecessarily bumps jobs to and from the much slower secondary storage. The net result is poor response time for all jobs running on the computer.

Some systems, such as the one described here, permit objects to be dynamically loaded. Only a small set of frequently used functions is actually part of the executable image. Other portions particular to a given task or implementation are bound as they are needed. This can greatly improve computer system performance and response time.

4 SYSTEM OVERVIEW

The system described here is a flexible and efficient computational tool for structural engineering applications. In this environment, not only may the problem specifications be easily modified, but new algorithms and techniques may also be readily implemented. Problem parameters may be monitored and changed. This system is based upon current available computer hardware such as an engineering workstation with a 32-bit processor, bit-mapped graphics display and computer networking interface. Besides being able to exploit the capabilities of this type of hardware, the system can also take advantage of other, more powerful machine architectures by linking tasks over a local area computer network.

This environment does not aim to replace existing structural engineering software, but rather to augment these programs by allowing the computer code to exist in a larger and more flexible framework. The system works in a manner similar to small, independent operating system built on top of the existing operating system generic to the workstation. In the spirit of many

other existing computer standards such as [32, 12], the current environment is described in terms of application layers. Other than a small and compact set of utilities which represent the core of the system, there is a great deal of latitude in what an individual implementation actually contains. In this way, new implementations can still compatibly exist with older ones and not be burdened by unnecessary details. For example, a particular implementation may not contain a computer network utility library because its application is not required. If, however, one is added at some later time, the guidelines are provided so that this library can interface and behave the same across all implementations.

The general structure of the environment is graphically depicted in Figure 1. There are four major components: the interpreter, a virtual machine, the set of structural support libraries and a package of applications libraries. Each of these components performs a specific and well defined task within the programming environment.

4.1 Virtual Machine

The virtual machine is the heart of the system. Providing the locus where computations are carried out, it is equivalent to a complete computer system implemented entirely in software. The virtual machine is constructed to provide an interface between the actual computer hardware along with its operating system software and the rest of the structural engineering programming environment. Not only does this arrangement provide a portable development platform for the rest of the modules, but it also localizes the changes that must be made when the entire environment is ported to a new computer system.

This virtual machine provides a mechanism for manipulating small pieces of data by executing simple operations. There is a segment to store instructions, a separate segment to store data and a few locations to keep temporary information which needs to be accessed quickly. Additionally, the machine usually operates by traversing a loop in which instructions are

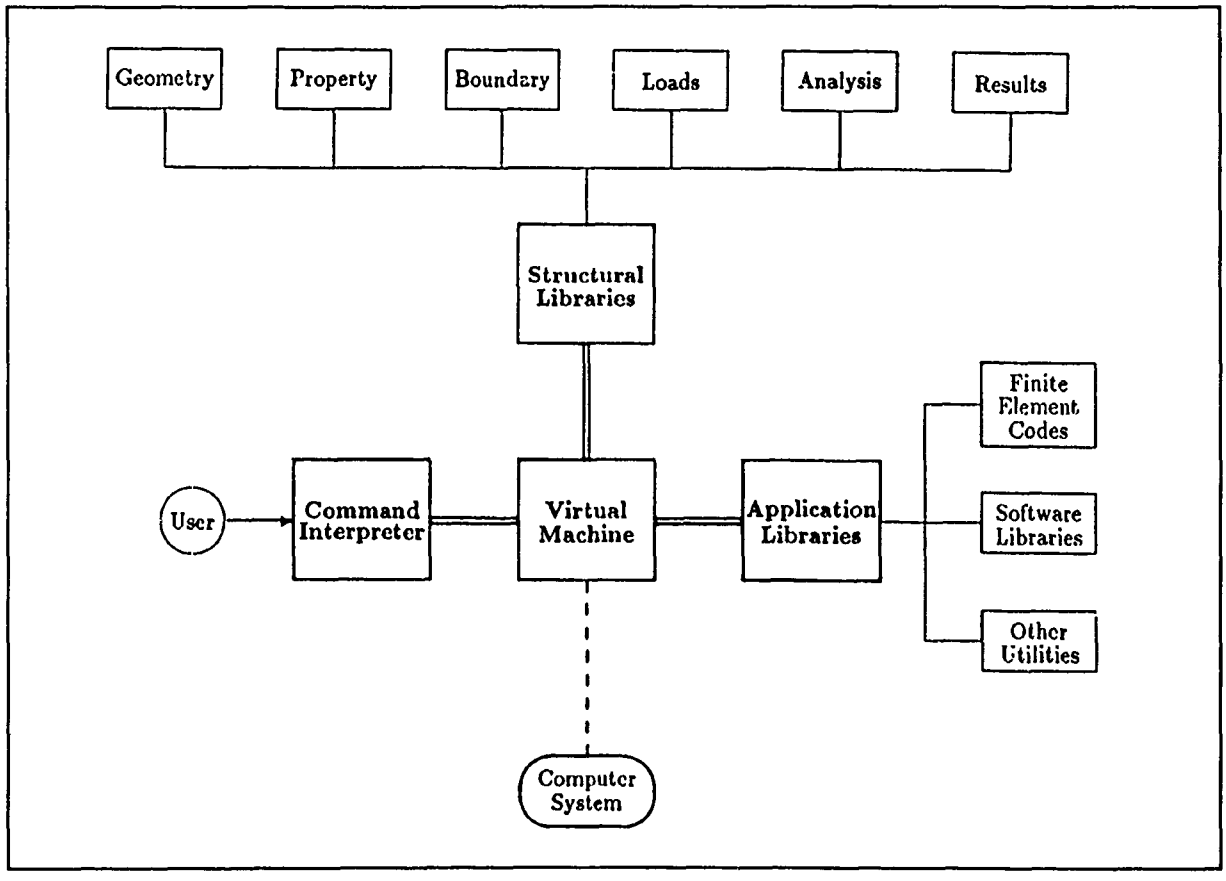


Figure 1: General structure of the programming environment.

decoded and executed in order to examine and modify information located in the data area of the system. This is completely analogous to the way in which the hardware inside a modern computer behaves. The separate data and instruction areas represent the physical memory of the computer, the temporary locations represent the machine registers, and the decoding of commands to examine and modify small pieces of data is the function of the central processor.

Some typical instructions used by the virtual machine are show in Table 1. Note that the only data types supported at this level are double precision floating point, integer and character or byte representations. All other data types, including those that are aggregated by the interpreter's programming language are decomposed into these simpler types by the parsing and code generation process.

The virtual machine described here is somewhat more complex than those discussed in [29] and [13]. This is not surprising since the task at hand is much more ambitious. For example, the input/output facilities are necessarily more robust. Furthermore, the system must interface to other software packages such as existing finite element codes or even subroutine libraries written in other languages. Hence, the machine must provide a mechanism for linking the virtual machine's data area with that of the externally provided module. To carry out this task, the machine provides the capability to map sections of its internal storage area onto the data area of other externally supplied subroutines and functions. The user can easily take advantage of this feature by using the rich variety of data types supplied by the interpreter's programming language.

Instruction	Parameters	Description
ALU	op, arguments	Binary arithmetic operations
DEC	region, size	Decrement value in region
INC	region, size	Increment value in region
LOD	region, size	Load data from region
STR	region, size	Store data to region
PSH	size	Push data onto the stack
POP	size	Pop data from the stack
JPZ	code, addres	Jump if code is zero
JMP	code, op, address	Compare code and Jump
CAL	address, arguments	Call local routine
FSL	address, arguments	Call remote routine

Table 1: Some typical instructions.

A simplified schematic of the virtual machine and how it interacts with the system's other major modules is shown in Figure 2. The box labeled "Control Logic" corresponds to the central processing unit, while the box labeled "Code Region" contains an internal representation of the user's program. The section labeled "Data Region" represents a simplification of the virtual machine's internal memory. Finally, the rectangle labeled "Foreign Data Mapping" corresponds to the capability of translating both data and code from external programs and libraries into a format that the virtual machine can understand.

Since the system provides support for exporting operations over a local area computer network, the machine must provide facilities to build and decode the portable data packets. This is critical because different kinds of computers may store objects in different ways. For example, even though most machines represent integers as 32 bit numbers, the underlying order of

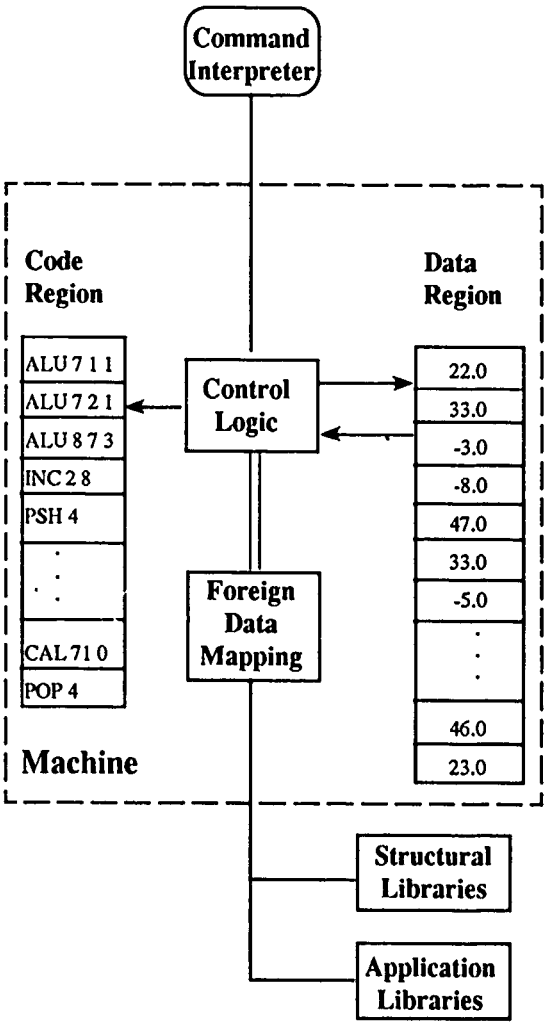


Figure 2: Virtual machine schematic.

these patterns may be quite different. The module labeled "Foreign Data Mapping" performs this translation, among others. It also must be able to bind to the network, establish connections and process the transaction.

There are several points that should be made about the implementation of the virtual machine. Most simple constructions use a stack arrangement. That is, the representation of the code and data regions depicted in Figure 2 are contiguous linear sections of memory. While this is the most straightforward approach, such an implementation may be somewhat slow. For this reason, the control logic contains some storage locations, corresponding to registers in an actual computer's hardware, where frequently used information can be stored. Also, since the virtual machine contains the lowest level functions in the entire programming environment which are frequently executed, pieces of this segment are frequently coded in assembly language. For example, the critical input/output facilities are very often highly dependent on the underlying computer hardware, so they are specially coded in assembly language.

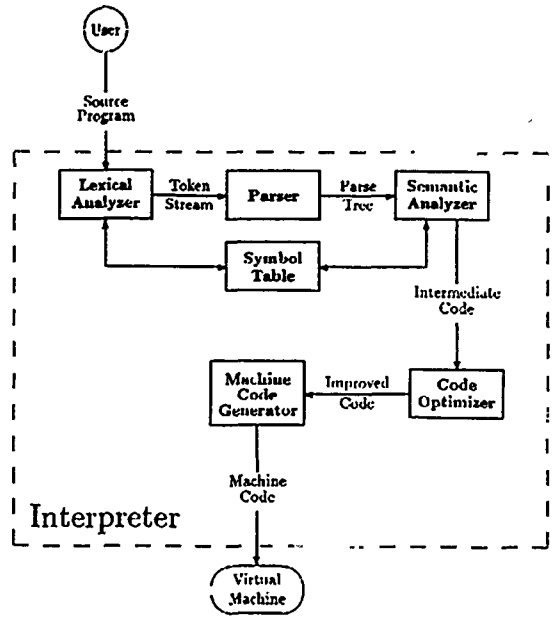


Figure 3: Command interpreter schematic.

4.2 Interpreter

The interpreter serves the critical function of translating the input into a form which can be efficiently used by the virtual machine. The structure of the language accepted by the programming environment plays a large part in design of this section. Supporting a generous number of program constructs and higher level functions, this module performs the translation of the user's input through several phases. Figure 3 schematically depicts the interpreter's operation.

The language constructs accepted by the interpreter are a mixture of both C and a dialect of **Fortran** similar to **Fortran 8x**. The language is structured and allows for user-defined data types. In addition to supporting functions written in its own language, a foreign function interface also exists for procedures written in other languages. Finally, objects may be dynamically bound to the programming environment.

Some typical programming constructs are shown in Table 2. These include a general purpose **if then else** statement, a **case** facility and several iteration or loop constructs.

Data types play an important part in this programming environment not only because they make development and implementation easier, but also because they facilitate the interaction between the software system and programs written externally. The basic building blocks of the data types of this system are shown in Table 3. Additional types may be mixed and combined with these. For example, a package of data representing a node might be constructed by combining floating point types with integers. Then, an array of 100 objects of this type might be declared as displayed below. Comments are enclosed in matching /* */'s.

```
type node {
   double x, y, z;      /* locations in space */
   integer fixity;      /* dx,dy,dz,rx,ry,rz */
   integer id;          /* a reference number */
};
node nodelist[100];  /* an array of 100 nodes */
```

Note that, in order to access the spatial location along the **X** axis of the fifth node, one would use:

```
nodelist[5].x
```

where the "." means "member of." This programming environment also has a limited pointer mechanism to allow portions of the virtual machines internal memory to be mapped to the storage locations of external data areas. A declaration of the form:

```
pointer(node,nodeptr)
```

would declare the variable **nodeptr** to be a pointer to an object of type **node**. The availability of this simple pointer type also means that dynamic memory alloca-

Statement Form	Description
var1 = *var2* = ... = *varN*	General assignment statement
{ *stmt1* ; *stmt2* ; ...; *stmtN* }	Statement block
if *cond* **then** *stmt* **else** *stmt*	If-then-else construct
case (*expr*) **case** *range* : *stmt* ...	Case selection
do (*iteration expression*) *stmt*	Iterated do loop
do (*expression*) *stmt*	Tested do loop
do (*value*) **times** *stmt*	Ranged do loop
var = *name* (*parameters*)	Function call
print (*format*, *expr1* ... *exprN*)	Print with format

Table 2: Some typical programming constructs.

Type	Size (in bytes)	Description
character	1	Smallest available unit
integer	4	General purpose integer values
double	8	General purpose floating point values
complex	16	Optionally implemented complex values
quad	18	Optionally implemented in floating point values

Table 3: Basic data types.

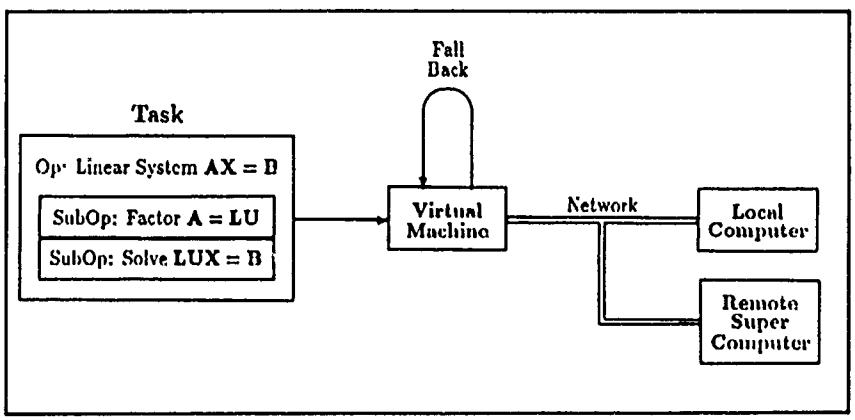


Figure 4: Remote procedure call schematic.

tion is possible within the system. Hence, one can allocate a list of objects of type **node** dynamically by using:

flag = allocate(nodeptr,node,100)

where the intrinsic function **allocate** is called to assign storage to the location pointed to by **nodeptr**. The flag variable returns an error status if **allocate** was unable to find sufficient space for the new data.

The foreign function interface provides access not only to the structural engineering and applications libraries written in other languages such as **C**, **LISP** or **Fortran**, but it also allows procedures to be accessed from across the local area network. This ability to export tasks to remote machines allows the programmer to dynamically select machine power for a given computational task with a fine level of control. Figure 4 schematically depicts the remote procedure call mechanism.

Network tasks are broken into operations at the procedure level. As shown in Figure 4, a sample operation such as the solution of a linear system of equations may be considered a single task, or it may be further divided into sub-tasks, such as factorization and forward- and back-substitution. In either case, the operation is dispatched to the virtual machine where a course of action is taken. The task may be executed locally by the virtual machine. Or, the operation might be done on a local computer. Alternatively, the task could be exported to a larger supercomputer if necessary. In the event that a given computer is not available, a fall-back mechanism is possible. Hence, a computation does not have to halt if access to remote machine is not possible, the calculation could fall-back to a local computer and continue processing. While the remote procedure call mechanism does entail a certain amount of overhead to set up and tear down network connections, it may still prove very worthwhile if the remote resources can provide sufficient capacity to handle very computationally intensive tasks.

Users may interact with the programming environment through the interpreter in three ways. The first is by directly entering commands into the system. The input is translated and run on the virtual machine synchronously. Any program information is buffered so that the user may return to the previously entered data and modify it by using a standard text editor. This mode of operation is similar to that provided by [9]. While this method of interaction can be very convenient for creating short programs or debugging algorithms, most larger problems often use a second or batch oriented method. Here, input is processed and executed by the virtual machine asynchronously. This method of operation not only gives a better level of performance for larger computations, but it also provides more opportunity for the optimization of numerical calculations. The third and final method of interaction requires the presence of a graphic application support library. With this method, some degree of interaction is provided for selecting objects in a display window. This last method is particularly valuable when computations make use of the structural support library.

As depicted in Figure 3, there are five major components to the command interpreter. Each of these modules serves to translate the user's input into a form suitable for execution on the virtual machine. Lexical analysis breaks the source program up into small pieces called tokens. Next, the parser constructs a new internal representation of the program as a tree. This key encoding preserves the program hierarchy and, through the symbol table, the form of the constants and variables in a portion of the software. During the third phase, the program is checked for semantic errors and some Type conversions are carried out. The code optimizer translates expression trees into directed graphs in order to find common expressions. Finally, machine code is generated during the last phase.

From an implementation standpoint, the interpreter represents the current state of the art in computer science applications. It makes use of an LALR parser [2], sophisticated code optimizer [15] and code generator [29]. In addition to providing facilities for interactive input and debugging support, the structural engineering programming environment described here can communicate over a local area network through the public domain RPC [6] mechanism.

Because the command interpreter plays such a critical role in how the system is used, there are many more features which could be added. A method for incremental compilation, such as the one described in [26] would greatly improve system response, although it might require that the form of the language be modified somewhat. Also, because different computers have substantially diverse ways of evaluating floating point expressions [20], a more general format is required for floating point values exported by the remote procedure mechanism. Unfortunately, this topic is still an active area of research and no standard method exists for dealing with this problem.

4.3 Structural Libraries

The structural libraries provide the primary means of applying the programming environment in engineering calculations. This collection of routines is specifically oriented toward solving structural engineering

problems. Beside providing a named work space for data, the code in this library supplies a collection of utilities for building and manipulating finite element models. A variety of standard structural analysis techniques is also available, algorithms and techniques which are not part of this library can be readily programmed by supplying information to the command interpreter.

This programming environment provides a storage area for structural engineering models. The major portions are listed and described in Table 4. There can be many different storage areas during a session, but each space has a unique name associated with it. Association between this named storage area and other portions of the system can be automatically mapped by the virtual machine.

Area	Description
Nodes	Spatial nodal locations
Elements	Element incidence pool
Materials	Geometrical and material parameters
Boundary	Displacement boundary conditions
Loads	Nodal and element force conditions

Table 4: Named work space data areas.

The structural support library separates engineering tasks into six distinct areas. This division roughly corresponds to the ways in which analytical models are commonly constructed, modified and used. Note that the libraries only provide an interface in terms of functions that can be called through the command interpreter. This collection of utilities is essentially stateless. When context information is required by one of the routines, an identification handle is returned by the routine which creates an instance of an object.

4.3.1 Geometry

The geometry subsection allows an engineer to build and change the geometrical description of a model. For example, this area contains finite element mesh generators for rectangular, cylindrical and spherical coordinate systems. In addition it may contain a graphical interface so that these meshes can be displayed on a terminal or hardcopy device. Some sample features of this package are listed in Table 5.

4.3.2 Property

The property subsection lets the engineer specify the constitutive and material properties of the model under consideration. This includes not only the physical properties such as mass, density and thickness but also analytical relationships such as yield surfaces, damage constraints and nonlinear response parameters. A sample of some of the features provided by this collection of routines is shown in Table 6.

Function	Description
gen grid (x1, y1, ... xN, yN, n, m)	Rectangular grid
move node (node, x1, y1, z1)	Move a node in space
id = add node (x1, y1, z1)	Add a node, return identifier
flag = delete node (node)	Delete a node, return code

Table 5: Some geometry operations.

Function	Description
id = add property (value1, ... valueN)	New instance
flag = change property (id, nth, value)	Change nth id entry
flag = delete property (id)	Delete a id, returning code

Table 6: Some property operations.

4.3.3 Boundary

Displacement constraints can be added or modified by using the utilities supplied in this package. In addition to providing simple support conditions, displacement constraints may also be specified. Such constraints might be useful during an analysis which requires either complex support conditions or involves the study of bodies subject to contact conditions. Some functions from this subsection are listed in Table 7.

Function	Description
id = add boundary (node)	New boundary instance
flag = delete boundary (id)	Delete support condition
flag = set boundary (id, code)	Add fixed support at id
flag = mast boundary (master, code, id)	Link id to master

Table 7: Some boundary operations.

4.3.4 Loads

By using this subsection of utilities, a user may specify the static and dynamic loadings on a model. Different loading cases are provided along with the capability to combine different loading conditions. Through the command interpreter, options are also available for generating loadings according to general functions. Several possible operations provided in this package are listed in Table 8.

Function	Description
id = add load (node, case)	New load instance for case
flag = delete load (id)	Delete load condition
id = combine load (id1, m1, id2, m2)	New load = $l1 * m1 + l2 * m2$
id = time load (kind, id, dt)	Dynamic load

Table 8: Some loading operations.

4.3.5 Analysis

This package of analysis routines allows the engineer to study the structural model by using a library of existing algorithms. Note that the programmer may also directly specify new algorithms and techniques directly at the interpreter level. This collection of software routines covers the most common analytical methods including static analysis, different approaches to solution of nonlinear systems, and options for dynamic analysis, including both time history and response spectrum solution methods. A few operations are listed in Table 9.

Function	Description
id = analysis feap (workspace)	Create a FEAP input file
flag = run feap (id)	Carry out a FEAP analysis
id = save feap (id)	Store FEAP response parameter

Table 9: Some analytical operations.

4.3.6 Results

This last collection of routines allows the engineer to extract information from the analysis phase in order to examine various response parameters. For example, end forces and moments might be converted to stresses or reactions may be calculated. Although all information in the analytical model's work space is available at the interpreter level, these routines provide a set of most common utilities usually required by the engineer in order to investigate the behavior of a structural model. Several features of this package are shown in Table 10.

Function	Description
disp = delta result (id)	Extract displacements
stress = stress result (id)	Extract element stresses
node = node result (id)	Project nodal stresses

Table 10: Features from the result package.

4.4. Application Libraries

The application libraries consist of a collection of utilities which, for the most part, optionally support the programming environment. They essentially provide a mechanism by which the system can be extended in a portable and compatible way. While the virtual machine, interpreter and structural libraries are necessary to the software system in terms of structural engineering calculations, the application libraries provide extra functionality.

By using the dynamic binding mechanism of the system, the programmer can make efficient use of the tools supplied by this collection of software. Figure 5 shows how new applications can "grow" to accommodate the new operations. Originally the two boxes on the left represent the original programming environment and a library of additional applications. The interpreter instructs the system to bind the new utility to the system dynamically. The box on the right represents a new version of the system, which includes the application. Once this binding has taken place, the user may call any function in the library as if it were originally part of the system. The layered approach is not only very efficient in today's virtual memory computers, but it also facilitates the construction of software systems tailored to specific applications and hardware environments. If a particular problem does not require a set of operations, then it never has to become part of the programming environment.

In the context of the present system, the applications library contains three separate utilities. They are: the system interface to a set of popular finite element codes, a mechanism for linking to some networking and mathematical libraries and a provision to connect to a network based computer graphics system.

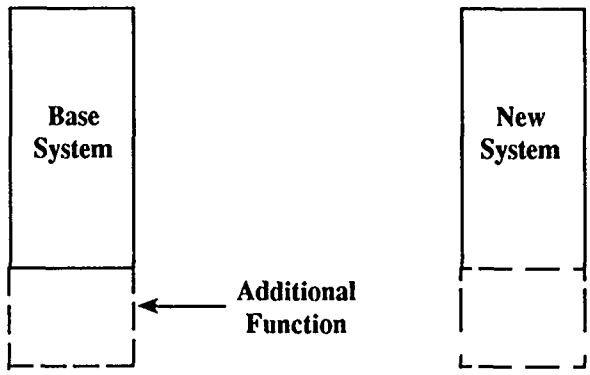


Figure 5: Dynamic binding.

The links to various finite element codes is usually system dependent. Many sites have definite preferences for the codes they wish to support and use. The library routines do not do any calculations themselves. Instead, they only implement the interface between the virtual machine and the data areas of the finite element programs.

The second set of utilities, access to network and mathematical libraries, performs two important tasks. The first is make available a set of functions which can map user level data objects into a portable network independent format. The second is provide an interface to several popular mathematical subroutine collections. These include software systems such as LINPACK [8], EISPACK [30], and the IMSL [10] collection of routines. This interface provides some powerful tools when new engineering algorithms are developed and studied.

The third group of application utilities is an interface to a graphics and workstation windowing system called X Windows [28]. The present library currently has the capability to display geometrical data, draw simple graphs and even animate the response of engineering models to various loading conditions.

There are many other possible applications which might be added in the future. For example, support for high level engineering graphics interface such as PHIGS [5], a library of routines for exploiting the parallel processing of capabilities of hardware, or a mechanism for translating the language accepted by the interpreter into standard **Fortran** would all be very useful additions to the programming environment.

5 EXAMPLES

In order to illustrate how this programming environment can be applied to structural engineering problems some short examples are listed here. The first example discusses how the system might be used to carry out a nonlinear analysis. Two other examples are presented by demonstrating the capabilities of the programming environment in common situations.

The task of studying the nonlinear response of a finite element system to a given loading condition involves three major steps. First, the model must be constructed. Next, a solution method must be chosen and applied to the model. Finally, response parameters are extracted and studied.

The construction of a finite element model in the system described in this report involves creating a named workspace for the data, generating information by entering commands to the interpreter and possibly examining a graphical representation of the structure by using the facilities provided by the window system. For instance, the following commands might be used to generate the geometrical description of the finite element model. The model is a 7 by 7 grid of elements 10 units by 10 units in size. The interior nodes are perturbed by some amount in order to study the behavior of a distorted mesh.

```
function my_geometric_model(model_id)
Model model_id;
{
integer i;
double random();
Node node_data;

model_id = new_model("Nonlinear Study");
gen_grid(0,0,10,0,10,10,0,10,7,7);
do i = 1,49 {
 node_data = node_info(i);
 if ((node_data.x > 0) and
  (node_data.y > 0) and
  (node_data.x < 10) and
  (node_data.y < 10)) then {
  /* random returns values in the range 0 < n < 1 */
  node_data.x = node_data.x + 0.5*(random() - 0.5;
  node_data.y = node_data.y + 0.5*(random() - 0.5);
  }
 }

return;

}
```

Loads, material properties, and boundary conditions are added and modified in a similar manner.

Next, an analysis technique is chosen. For simple nonlinear behavior, the engineer might elect to study the system using an existing finite element package. Hence, the user could simply generate an input file for the code.

```
generate_input(model_id)
Model model_id;
{
   /* 10 steps of newton raphson iteration */
   describe_analysis(model_id,"newton",10);
   /*"code" is the package which will use the analysis */
   gen_code(model_id,"code");
   return
  }
```

Alternatively, the user might specify the algorithm directly, instead of using the templates provided by the system. For example, to perform 10 Newton-Raphson iterations, the following commands could be used.

```
my_newton(model_id)
Model model_id;
{
  do (10) times {
  make-tangent(model_id);
  make_residual(model_id);
  get_displacements(model_id);
  }
}
```

Other functions are invoked to construct and assemble the model's stiffness and load matrices.

Finally, the response parameters corresponding to displacements and stress may be examined. The response parameters may be combined with the mesh geometry, or they be output directly as numerical values.

Two other demonstration examples are also shown here. Both of these seek to illustrate some of the working capabilities of the current programming environment.

The first example is a demonstration system for investigating the response of a single degree of freedom system. In order to carry out the required calculations, the programming environment interfaces to the MACSYMA [21] symbolic manipulation system. Using this tool, an analytical result in terms of the model's physical parameters k, m and c may be obtained. The MACSYMA system generates a result in the **Fortran** language, which is then returned to the programming environment. Given this information, a graphical representation of the system may be animated to show how the system might respond to a given set of initial conditions.

Figure 6 shows a copy of the workstation's display during the animation process. Three windows appear on the screen. The bottom window contains the programming environment's command interpreter. Note that the lines input by the user are numbered, so that they may later be recalled, if necessary. The short portion of program text visible in the window is representative of how a piece of software would appear for any application. The window in the upper right hand corner contains a script for the MACSYMA system. This collection of commands instructs MACSYMA to symbolically solve the differential equation of motion of the simple model. Only a portion of the entire script is visible in this window. The window in the upper left hand corner depicts the physical representation of the model, and how it responds to various excitations.

This example illustrates three important features of the system. First, it shows the environment's capability to communicate with other computer programs, even those written in different languages. Second, the example clarifies the relationship between the command interpreter, network interface and virtual machine by placing each of the operations in a separate window. Finally, it also features the graphic capabilities of the environment.

The second example illustrates the use of the system for the study of the nonlinear response of a braced frame due to an earthquake excitation. Here, a six story frame is subjected to a scaled ground acceleration. The actual analysis is carried out over a local area computer network, and selected results are saved for later study.

Four windows are depicted in Figure 7. As before, the bottom window contains the programming environment's command interpreter. The particular segment of code visible in the window scales the displacement response values so that they may be more readily seen in the graphic display. The windows at the top of the screen contain the animated response of the frame model. The window at the top right illustrates the entire model and its response to the loading. Note that brace on the second level has buckled and undergone a permanent vertical deformation. The windows in the top right hand portion of the screen are attached to specific response values. In this case, these happen to be the top story horizontal displacement and the second story vertical displacement. Of course, other monitoring is possible.

This example illustrates the capabilities of the system to exploit a local area computer network connection in order to carry out an analysis and forward the results back to the programming environment. Furthermore, it also shows the system's ability to integrate the analysis operation with a workstation's graphic display.

6 SUMMARY

This report gives a somewhat detailed overview of a programming environment for structural engineering computations. While the system is primarily intended

for the research community, it may also find a wide variety of applications in commercial production situations as well. The programming system is highly modularized, provides a flexible and extensible platform for software development and highly suited to today's computer technology.

With the availability of low cost engineering workstations, equipped with powerful central processors, a computer network interface and high resolution computer graphic capabilities, the computational opportunities for structural engineers is gradually changing. No longer does the development of new algorithms and techniques require that a programmer carry out many iterations of the "edit, compile and debug" cycle. New computer hardware has changed all of that. It is now possible to provide a custom environment for structural engineering applications which provides support not only in terms of the creation and manipulation of finite element models, but also in the development and study of new algorithms and computational strategies.

The system described in this report is intended to provide state of the art facilities for engineering calculations. Accordingly, this programming environment contains several innovative ideas. Some of these are listed below:

- The system is compartmentalized into four major areas: an interpreter, the virtual machine, a set of structural libraries and a collection of application libraries. This division not only makes the development and maintenance of the system much more straightforward, but it also makes the environment much easier to use by providing a logical overall framework.

- A programmable and extensible command interpreter is the major user level interface to the environment. The command language is quite sophisticated. It allows the engineer to write loops, conditionals, internal procedures and define variables. Furthermore, the system has several basic data types, and new data types may be defined by the programmer.

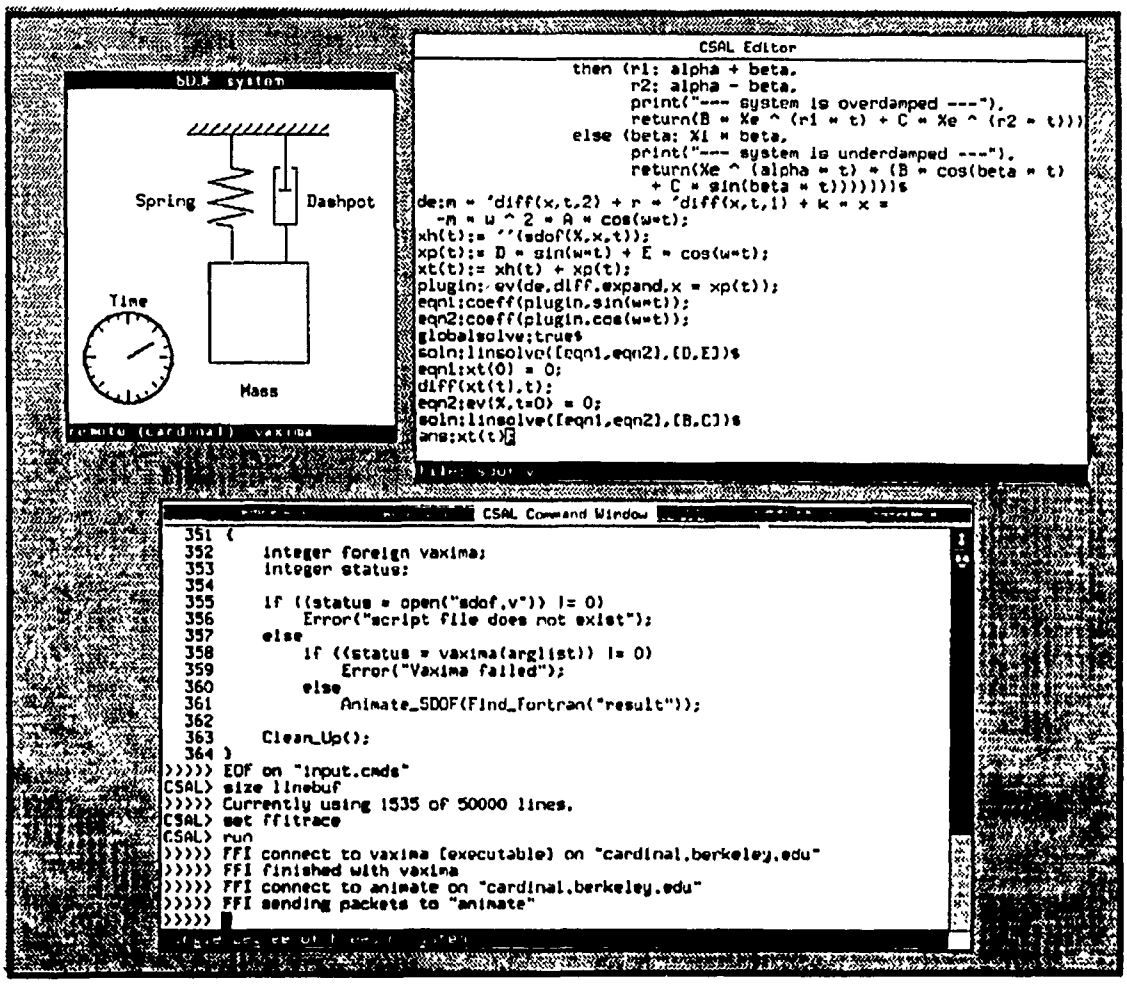


Figure 6: Single degree of freedom demonstration.

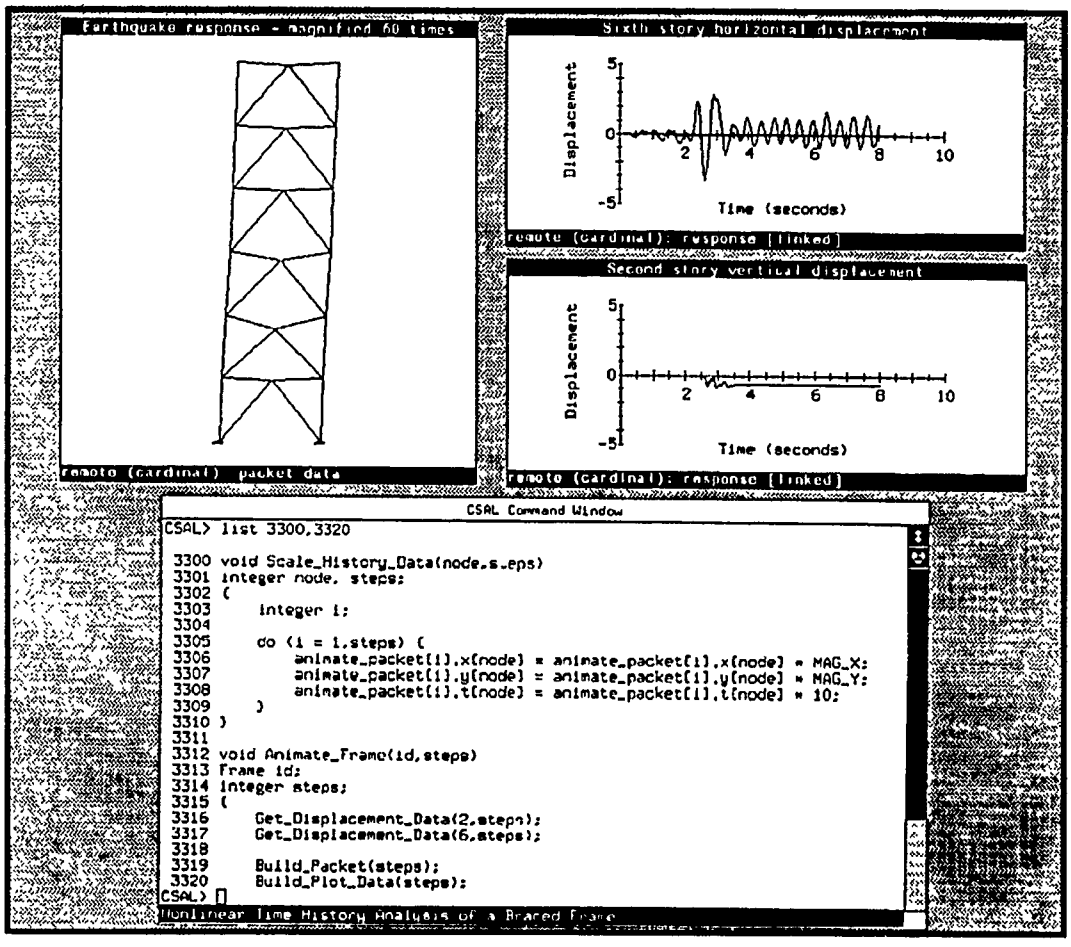


Figure 7: Nonlinear response of a braced frame.

• This environment supports computer network operations. In particular, tasks may be exported to different computers across a local area network on a procedure basis. In addition to providing a fall back mechanism if remote resources are not available, the system will automatically convert data to and from a computer independent representation.

• A virtual machine makes the system easier to move to new computers and operating systems by localizing the areas where changes need to be made. The virtual machine provides a level of abstraction above the actual workstation hardware and software, and it helps to make the various system interfaces much more uniform.

• By allowing objects to be bound to the system at run time, this computer environment can efficiently run on today's engineering workstations. Dynamic binding not only makes the base system much smaller, but it also allows library routines to be developed asynchronously.

• The programming environment is comprised of several layers of implementation. Only a small collection of utilities makes up the basic system, all other items may be optionally added if they are required by an implementation.

• This system provides a callable interface to a structural support library. A collection of routines in this library, along with a named work space, allows an engineer to flexibly tailor how a given engineering model might be represented. Furthermore, since algorithms can also be described at the command interpreter level, there is no longer a division between a finite element model and the operations performed on it. Hence, for example, new techniques such as adaptive h and p order mesh refinement may be more readily implemented.

• The set of application libraries can be used to take advantage of the large body of existing computer code. For example, the system has the capability to communicate with large finite element codes and a set of standard mathematical libraries.

This programming environment for structural engineering applications can offer a great deal of support to the users. Because the system is primarily interactive, the engineer can examine and modify data and algorithms quite easily. By taking advantage of several emerging standards for computer interaction, graphics and networking, this programming environment can be constructed with today's technology.

7 ACKNOWLEDGMENT

Supervision and guidance of this study were provided by Professor R. L. Taylor, University of California, Berkeley.

8 REFERENCES

[1] Ada Joint Program Office (1983) *Reference Manual for the Ada Programming Language*, ANSI/Military Standard MIL-STD-1815A, Washington, D.C.; United States Department of Defense.

[2] Aho, A.V.; Sethi, R.; Ullman, J.D. (1986) *Compilers, Principles, Techniques and Tools*, Reading, Mass.; Addison-Wesley.

[3] ANS X3.9 (1978) *American National Standard Programming Language Fortran*, New York; American National Standards Institute.

[4] ANS X3.9-198x (1987) *Draft Proposed Revised American National Standard Programming Language Fortran*, New York; American National Standards Institute.

[5] ANS X3H3/85-21 (1985) *Draft American National Standard for the Functional Specification of the Programmer's Hierarchical Interactive Graphics System (PHIGS)*, New York; American National Standards Institute.

[6] Birrell, A.D.; Nelson, B.J. (1984) *Implementing Remote Procedure Calls*, ACM Transactions on Computer Systems, 2.1, 39-59.

[7] DeSalvo, G.J.; Swanson, J.A. (1985) *ANSYS User's Manual*, Houston, Pa.; Swanson Analysis Systems, Inc.

[8] Dongarra, J.J.; Moler, C.B.; Bunch, J.R.; Stewart, G.W. (1979) *LIN-PACK User's Guide*, Philadelphia, P.A.; SIAM.

[9] Goldberg, A.; Robson, D. (1983) *Smalltalk-80: The Language and Its Implementations*, Reading Mass.; Addison-Wesley.

[10] IMSL, Inc; (1982) *IMSL Library Reference Guide*, Edition 9, Houston, Texas; IMSL, Inc.

[11] Institute of Electrical and Electronics Engineers (1987) *Guide to POSIX Based Open System Architecture*, Washington, DC; IEEE Computer Standards Secretariat.

[12] International Standards Organization (1981) *Graphical Kernel System (GKS)*, Version 6.6.

[13] Kernighan, B.W. and Pike, R. (1984) *The Unix Programming Environment*, Englewood Cliffs, NJ; Prentice Hall.

[14] Landers, J.A.; Austin, M.A.; Taylor, R.L.; Pister, K.S (1986) *A Programming Environment for Structural Engineering Computations*, Proceedings of the First World Conference on Computational Mechanics, Austin, Texas.

[15] Landers, J.A. (to appear) *PGF: A Postprocessor for generated Fortran*, Engineering with Computers.

[16] Landers, J.A. (1982) *SDMS: The Structural Data Management System*, Chicago, Ill.; Skidmore, Owings and Merrill.

[17] Logcher, R.D. (1967) *ICES STRUDL-I The Structural Design Language*, Cambridge Mass.; MIT Department of Civil Engineering.

[18] Lopez, L.A. (1972) *POLO: Problem-Oriented Language Organizer*, Computers and Structures, 2.4, 555-572.

[19] MacNeal, R.H.; McCormick, C.W. (1971), *The NASTRAN Computer Program for Structural Analysis*, Computers and Structures, 1.3, 389-412.

[20] Maguire, G.Q; Smith, J.M. (1988) *Process Migration: Effects on Scientific Computation*, SIGPLAN Notices, 23.3, 102-106.

[21] MATHLAB Group (1977) *MACSYMA Reference Manual, Version Ten*, Cambridge, Mass.; MIT Laboratory for Computer Science.

[22] McDonnell Douglas Automation Company (1974) *ICES STRUDL Improvements User's Manual*, Technical Note M1090043.

[23] Moler, C. (1982) *MATLAB User's Guide*, Department of Computer Science, University of New Mexico.

[24] Parisi, M.A.; Rehak, D.R. (1986) *General Purpose Software for Probability Computations - A Virtual Machine Approach*, Engineering with Computers. 1.3, 61-173.

[25] PDA Engineering, *PATRAN User's Guide: Volumes 1-2*, Santa Ana, CA; PDA Engineering.

[26] Reps, T.W. (1984) *Generating Language-Based Environments*, Cambridge, Mass.; MIT Press.

[27] Sandberg, R. (1985) *Design and Implementation of the Sun Network Filesystem*, Proceedings of the Usenix 1985 Summer Conference, 119-130.

[28] Scheifler, R.; Gettys, J. (1986) *The X Window System*, LCS Memo LCS-TM-368, Cambridge Mass.: MIT Laboratory for Computer Science.

[29] Schreiner, A.T.; Friedman, H.G. (1985) *Introduction to Compiler Construction with UNIX*, Englewood Cliffs, NJ; Prentice Hall.

[30] Smith, B.T.; Boyle, J.M.; Dongarra, J.J.; Garbow, B.S.; Ikebe, Y.; Klema, V.C.; Moler; C.B. (1976) *Matrix Eigensystem Routines - EIS-PACK Guide*, New York; Springer-Verlang Lecture Notes in Computer Science.

[31] SOFTECH Microsystems (1980) *UCSD Pascal User's Manual*, San Diego, CA.

[32] SRI International (1985) *1985 DDN Protocol Handbook*, Menlo Park, CA; DDN Network Information Center.

[33] Taylor, R.L. (1977) *Computer Procedures for Finite Element Analysis, in The Finite Element Method* by O.C. Zienkiewicz, London: McGraw-Hill.

[34] Wilensky, R. (1984) *LISPcraft*, New York; W.W. Norton.

[35] Wilson, E.L. (1979) *CAL 78 User Information Manual*, SESM Report Number 79-1, University of California, Berkeley; Department of Civil Engineering

[36] Wilson, E.L. (1970) *A General Structural Analysis Program (SAP)*, SESM Report Number 70-20, University of California, Berkeley; Department of Civil Engineering.

[37] Wirth, N. (1976) *Algorithms + Data Structures = Programs*, Englewood Cliffs, NJ; Prentice Hall.

DISTRIBUTION LIST

AFESC TIC (library), Tyndall AFB, Fl
ARMY CECOM R&D Tech Lib, Ft Monmouth, NJ
ARMY BELVOIR R&D CEN STRBE-JB, Ft Belvoir, VA
ARMY CERL Library, Champaign, IL
ARMY ENGR DIST Library, Seattle, WA; Phila, Lib, Philadelphia, PA
ARMY EWES Library, Vicksburg MS
ARMY LMC Fort Lee, VA
ARMY MMRC DRXMR-SM (Lenoe), Watertown, MA
ASST SECRETARY OF THE NAVY RE&S, Washington, DC
CBC Tech Library, Gulfport, MS
CBU 403, OIC, Annapolis, MD
CNA Tech Library, Alexandria, VA
COMDT COGARD Library, Washington, DC
DIA DB-6E1, Washington, DC
DIRSSP Tech Lib, Washington, DC
DNA Tech Svcs Lib, Mercury, NV
DOD Explos Safety Brd (Lib), Washington, DC
DOE Knolls Atomic Pwr Lab, Lib, Schenectady, NY; Wind/Ocean Tech Div, Tobacco, MD
DTIC Alexandria, VA
GIDEP OIC, Corona, CA
GSA Ch Engrg Br, PQB, Washington, DC
LIBRARY OF CONGRESS Sci & Tech Div, Washington, DC
NAVCOASTSYSCEN Tech Library, Panama City, FL
NAVCOMMSTA Library, Diego Garcia
NAVEODTECHCEN Tech Library, Indian Head, MD
NAVFACENGCOM Code 09M124 (Lib), Alexandria, VA
NAVFACENGCOM - CHES DIV, FPO-1PL, Washington, DC
NAVFACENGCOM - NORTH DIV, Code 04AL, Philadelphia, PA
NAVFACENGCOM - PAC DIV, Library, Pearl Harbor, HI
NAVFACENGCOM - SOUTH DIV, Library, Charleston, SC
NAVFACENGCOM - WEST DIV, Code 04A2.2 (Lib), San Bruno, CA
NAVOCEANSYSCEN Code 9642B, San Diego, CA
NAVPGSCOL Code 69 (T. Sarpkaya), Monterey CA
NAVSCOLCECOFF Code C35, Port Hueneme, CA
NAVSHIPREPFAC Library, Guam
NAVSHIPYD Code 202.5 (Library), Bremerton, WA; Library, Portsmouth, NH
NAVWARCOL Code 24, Newport, RI
NRL Code 4670 (B. Faraday), Washington, DC
NTIS Lehmann, Springfield, VA
NUSC DET Lib (Code 4533), Newport, RI
OCNR Code 1113, Arlington, VA
OFFICE OF SECRETARY OF DEFENSE DDR&E, Washington, DC
PMTC Code 1018, Point Mugu, CA
PWC Code 123C, San Diego, CA; Library (Code 134), Pearl Harbor, HI; Library, Guam, Mariana Islands; Library, Norfolk, VA; Library, Pensacola, FL; Library, Yokosuka, Japan; Tech Library, Subic Bay, RP
SUPSHIP Tech Library, Newport News, VA
US GOVT PRINTING OFFC Library Prgms Svcs, SLLC, Washington, DC; Supt Docs, SLLA, Washington, DC
USNA Ch, Mech Engrg Dept, Annapolis, MD; Ocean Engrg Dept (McCormick), Annapolis, MD
CALIFORNIA STATE UNIVERSITY C.V. Chelapati, Long Beach, CA
CASE WESTERN RESERVE UNIV CE Dept (Perdikaris), Cleveland, OH
CATHOLIC UNIV of Am, CE Dept (Kim), Washington, DC
CITY OF LIVERMORE Dackins, PE, Livermore, CA
CLARKSON COLL OF TECH CE Dept (Batson), Potsdam, NY
COLORADO STATE UNIVERSITY CE Dept (Criswell), Ft Collins, CO
CORNELL UNIVERSITY Civil & Environ Engrg (Dr. Kulhawy), Ithaca, NY; Library, Ithaca, NY
DAMES & MOORE Library, Los Angeles, CA
FLORIDA ATLANTIC UNIVERSITY Ocean Engrg Dept (Martin), Boca Raton, FL; Ocean Engrg Dept (Su), Boca Raton, FL
FLORIDA INST OF TECH CE Dept (Kalajian), Melbourne, FL
GEORGE WASHINGTON UNIV Engrg & App Sci Scol (Fox), Washington, DC
GEORGIA INSTITUTE OF TECHNOLOGY CE Scol (Kahn), Atlanta, GA; CE Scol (Swanger), Atlanta, GA; CE Scol (Zuruck), Atlanta, GA
INSTITUTE OF MARINE SCIENCES Library, Port Aransas, TX
JOHNS HOPKINS UNIV CE Dept (Jones), Baltimore, MD
LAWRENCE LIVERMORE NATL LAB FJ Tokarz, Livermore, CA; Plant Engrg Lib (L-654), Livermore, CA

LEHIGH UNIVERSITY Linderman Library, Bethlehem, PA
LONG BEACH PORT Engrg Dir (Allen), Long Beach, CA
MICHIGAN TECH UNIVERSITY CE Dept (Haas), Houghton, MI
MIT Engrg Lib, Cambridge, MA; Lib, Tech Reports, Cambridge, MA
NATL ACADEMY OF SCIENCES NRC, Naval Studies Bd, Washington, DC
OKLAHOMA STATE UNIV Ext Dist Offc, Tech Transfer Cen, Ada OK
OREGON STATE UNIVERSITY CE Dept (Hicks), Corvallis, OR
PENNSYLVANIA STATE UNIVERSITY Gotolski, University Park, PA; Rsch Lab (Snyder), State College, PA
PORTLAND STATE UNIVERSITY Engrg Dept (Migliore), Portland, OR
PURDUE UNIVERSITY CE Scol (Chen), W. Lafayette, IN; CE Scol (Leonards), W. Lafayette, IN; Engrg Lib, W. Lafayette, IN
SAN DIEGO STATE UNIV CE Dept (Krishnamoorthy), San Diego, CA
SEATTLE PORT W Ritchie, Seattle, WA
SEATTLE UNIVERSITY CE Dept (Schwaegler), Seattle, WA
SOUTHWEST RSCH INST Energetic Sys Dept (Esparza), San Antonio, TX; King, San Antonio, TX; M. Polcyn, San Antonio, TX; Marchand, San Antonio, TX
STATE UNIVERSITY OF NEW YORK CE Dept (Reinhorn), Buffalo, NY; CE Dept, Buffalo, NY
TEXAS A&M UNIVERSITY CE Dept (Machemehl), College Station, TX; CE Dept (Niedzwecki), College Station, TX; Ocean Engr Proj, College Station, TX
UNIVERSITY OF CALIFORNIA CE Dept (Fenves), Berkeley, CA; CE Dept (Fourney), Los Angeles, CA; CE Dept (Gerwick), Berkeley, CA; CE Dept (Polivka), Berkeley, CA; CE Dept (Williamson), Berkeley, CA; Naval Archt Dept, Berkeley, CA
UNIVERSITY OF HARTFORD CE Dept (Keshawarz), West Hartford, CT
UNIVERSITY OF HAWAII CE Dept (Chiu), Honolulu, HI; Manoa, Library, Honolulu, HI; Ocean Engrg Dept (Ertekin), Honolulu, HI
UNIVERSITY OF ILLINOIS Library, Urbana, IL; Metz Ref Rm, Urbana, IL
UNIVERSITY OF MICHIGAN CE Dept (Richart), Ann Arbor, MI
UNIVERSITY OF MISSOURI Military Sci Dept, Rolla, MO
UNIVERSITY OF NEBRASKA Polar Ice Coring Office, Lincoln, NE
UNIVERSITY OF NEW MEXICO HL Schreyer, Albuquerque, NM; NMERI (Bean), Albuquerque, NM; NMERI (Falk), Albuquerque, NM; NMERI (Leigh), Albuquerque, NM
UNIVERSITY OF PENNSYLVANIA Dept of Arch (P. McCleary), Philadelphia, PA
UNIVERSITY OF RHODE ISLAND CE Dept (Kovacs), Kingston, RI; CE Dept, Kingston, RI
UNIVERSITY OF TEXAS CE Dept (Thompson), Austin, TX; Construction Industry Inst, Austin, TX; ECJ 4.8 (Breen), Austin, TX; Fusion Studies Inst (Kotschenreuther), Austin, TX
UNIVERSITY OF WASHINGTON CE Dept (Hartz), Seattle, WA; CE Dept (Mattock), Seattle, WA
UNIVERSITY OF WISCONSIN Great Lakes Studies Cen, Milwaukee, WI
WASHINGTON OES/PHS/DDHS (Ishihara), Seattle, WA
ADVANCED TECHNOLOGY, INC Ops Cen Mgr (Bednar), Camarillo, CA
AMERICAN CONCRETE INSTITUTE Library, Detroit, MI
ARCHITECTURAL STUDIO 3 M Mrvos, Long Beach, CA
ARVID GRANT & ASSOC Olympia, WA
ATLANTIC RICHFIELD CO RE Smith, Dallas, TX
BATTELLE D Frink, Columbus, OH
BECHTEL CIVIL, INC K. Mark, San Francisco, CA; Woolston, San Francisco, CA
BETHLEHEM STEEL CO Engrg Dept (Dismuke), Bethlehem, PA
BRITISH EMBASSY Sci & Tech Dept (Wilkins), Washington, DC
BROWN & ROOT Ward, Houston, TX
CHEVRON OIL FLD RSCH CO Strickland, La Habra, CA
CHILDS ENGRG CORP K.M. Childs, Jr, Medfield, MA
CLARENCE R JONES, CONSULTN, LTD Augusta, GA
COLLINS ENGRG, INC M Garlich, Chicago, IL
CONRAD ASSOC Luisoni, Van Nuys, CA
CONSOER TOWNSEND & ASSOC Schramm, Chicago, IL
CONSTRUCTION TECH LABS, INC G. Corley, Skokie, IL
CURTIS ENGRG CORP DH Curtis, National City, CA
DAVY DRAVO Wright, Pittsburg, PA
DILLINGHAM CONSTR CORP (HD&C), F McHale, Honolulu, HI
EARL & WRIGHT CONSULTING ENGRGS Jensen, San Francisco, CA
EVALUATION ASSOC, INC MA Fedele, King of Prussia, PA
GRIDCO Ong Yam Chai, Singapore
GRUMMAN AEROSPACE CORP Tech Info Ctr, Bethpage, NY
GULF COAST RSCH LAB Library, Ocean Springs, MS

ADINA ENGRG, INC / Walczak, Watertown, MA
AFOSR / NA (LT COL L.D. Hokanson), Washington, DC
APPLIED RSCH ASSOC, INC / Higgins, Albuquerque, NM
ARMSTRONG AERO MED RSCH LAB / Ovenshire, Wright-Patterson AFB, OH
ARMY CORPS OF ENGRS / HQ, DAEN-ECE-D (Paavola), Washington, DC
ARMY EWES / WES (Norman), Vicksburg, MS
ARMY EWES / WES (Peters), Vicksburg, MS
ARMY EWES / WESIM-C (N. Radhadrishnan), Vicksburg, MS
CATHOLIC UNIV / CE Dept (Kim) Washington, DC
CENTRIC Engineering Systems, Inc / Taylor, Palo Alto, CA
DOT / Transp Sys Cen (Tong), Cambridge, MA
DTIC / Alexandria, VA
DTRCEN / (Code 1720), Bethesda, MD
GEN MOTORS RSCH LABS / (Khalil), Warren, MI
GEORGIA INST OF TECH / Mech Engrg (Fulton), Atlanta, GA
HQ AFESC / RDC (Dr. M. Katona), Tyndall AFB, FL
LOCKHEED / Rsch Lab (B. Nour-Omid), Palo Alto, CA
LOCKHEED / Rsch Lab (M. Jacoby), Palo Alto, CA
LOCKHEED / Rsch Lab (P. Underwood), Palo Alto, CA
LOCKHEED / Rsch Lab (S. Nour-Omid), Palo Alto, CA
MARC ANALYSIS RSCH CORP / Hsu, Palo Alto, CA
MEDWADOWSKI, S. J. / Consult Struct Engr, San Francisco, CA
NAVFACENGCOM / Code 04B2 (J. Cecilio), Alexandria, VA
NAVFACENGCOM / Code 04BE (Wu), Alexandria, VA
NORTHWESTERN UNIV / CE Dept (Belytschko), Evanston, IL
NRL / Code 4430, Washington, DC
NSF / Struc & Bldg Systems (KP Chang), Washington, DC
NUSC DET / Code 44 (Carlsen), New London, CT
OCNR / Code 10P4 (Kostoff), Arlington, VA
OCNR / Code 1121 (EA Silva), Arlington, VA
OCNR / Code 1132SM, Arlington, VA
OHIO STATE UNIV / CE Dept (Sierakowski), Columbus, OH
OREGON STATE UNIV / CE Dept (Hudspeth), Corvallis, OR
OREGON STATE UNIV / CE Dept (Leonard), Corvallis, OR
OREGON STATE UNIV / CE Dept (Yim), Corvallis, OR
OREGON STATE UNIV / Dept of Mech Engrg (Smith), Corvallis, OR
PORTLAND STATE UNIV / Engrg Dept (Migliori), Portland, OR
SRI INTL / Engrg Mech Dept (Grant), Menlo Park, CA
SRI INTL / Engrg Mech Dept (Simons), Menlo Park, CA
STANFORD UNIV / App Mech Div (Hughes), Stanford, CA
STANFORD UNIV / CE Dept (Pensky), Stanford, CA
STANFORD UNIV / Div of App Mech (Simo), Stanford, CA
TRW INC / Crawford, Redondo Beach, CA
TRW INC / Dr. N. Carpenter, San Bernardino, CA
UNIV OF CALIFORNIA / CE Dept (Herrmann), Davis, CA
UNIV OF CALIFORNIA / CE Dept (Kutter), Davis, CA
UNIV OF CALIFORNIA / CE Dept (Romstad), Davis, CA
UNIV OF CALIFORNIA / CE Dept (Shen), Davis, CA
UNIV OF CALIFORNIA / CE Dept (Wilson), Berkeley, CA
UNIV OF CALIFORNIA / Ctr for Geotech Model (Idriss), Davis, CA
UNIV OF CALIFORNIA / Geotech Model Cen (Cheney), Davis, CA
UNIV OF CALIFORNIA / Mech Engrg Dept (Bayo), Santa Barbara, CA

HALEY & ALDRICH, INC. T.C. Dunn, Cambridge, MA
HAYNES & ASSOC H. Haynes, PE, Oakland, CA
HIRSCH & CO L Hirsch, San Diego, CA
HJ DEGENKOLB ASSOC W Murdough, San Francisco, CA
HOPE ARCHTS & ENGRS San Diego, CA
HUGHES AIRCRAFT CO Tech Doc Cen, El Segundo, CA
INTL MARITIME, INC D Walsh, San Pedro, CA
IRE-ITTD Input Proc Dir (R. Danford), Eagan, MN
JOHN J MC MULLEN ASSOC Library, New York, NY
LEO A DALY CO Honolulu, HI
LIN OFFSHORE ENGRG P. Chow, San Francisco CA
LINDA HALL LIBRARY Doc Dept, Kansas City, MO
MARATHON OIL CO Gamble, Houston, TX
MARITECH ENGRG Donoghue, Austin, TX
MC CLELLAND ENGRS, INC Library, Houston, TX
MOBIL R&D CORP Offshore Engrg Lib, Dallas, TX
MT DAVISSON CE, Savoy, IL
EDWARD K NODA & ASSOC Honolulu, HI
NEW ZEALAND NZ Concrete Rsch Assoc, Library, Porirua
NORTHWEST ENGRG CO Grimm, Bellevue, WA
NUHN & ASSOC A.C. Nuhn, Wayzata, NM
PACIFIC MARINE TECH (M. Wagner) Duvall, WA
PILE BUCK, INC Smoot, Jupiter, FL
PMB ENGRG Coull, San Francisco, CA
PORTLAND CEMENT ASSOC AE Fiorato, Skokie, IL
PRESNELL ASSOC, INC DG Presnell, Jr, Louisville, KY
SANDIA LABS Library, Livermore, CA
SARGENT & HERKES, INC JP Pierce, Jr, New Orleans, LA
SAUDI ARABIA King Saud Univ, Rsch Cen, Riyadh
SEATECH CORP Peroni, Miami, FL
SHELL OIL CO E Doyle, Houston, TX
SIMPSON, GUMPERTZ & HEGER, INC E Hill, CE, Arlington, MA
3M CO Tech Lib, St. Paul, MN
TRW INC Crawford, Redondo Beach, CA; Dai, San Bernardino, CA; Engr Library, Cleveland, OH; Rodgers, Redondo Beach, CA
TUDOR ENGRG CO Ellegood, Phoenix, AZ
VSE Ocean Engrg Gp (Murton), Alexandria, VA
VULCAN IRON WORKS, INC DC Warrington, Cleveland, TN
WESTINGHOUSE ELECTRIC CORP Library, Pittsburg, PA
WISS, JANNEY, ELSTNER, & ASSOC DW Pfeifer, Northbrook, IL
WISWELL, INC G.C. Wiswell, Southport, SC
WOODWARD-CLYDE CONSULTANTS West Reg, Lib, Oakland, CA
BROWN, ROBERT University, AL
BULLOCK, TE La Canada, CA
CHAO, JC Houston, TX
CLARK, T. Redding, CA
CURTIS, C. Ventura, CA
DOBROWOLSKI, JA Altadena, CA
GIORDANO, A.J. Sewell, NJ
HARDY, S.P. San Ramon, CA
HAYNES, B. No. Stonington, CT
HEUZE, F Alamo, CA
KOSANOWSKY, S Pond Eddy, NY
NIEDORODA, AW Gainesville, FL
PETERSEN, CAPT N.W. Pleasanton, CA
QUIRK, J Panama City, FL
SMELSER, D Sevierville, TN
SPIELVOGEL, L Wyncote, PA
STEVENS, TW Dayton, OH
VAN ALLEN, B Kingston, NY

UNIV OF CALIFORNIA / Mech Engrg Dept (Bruch), Santa Barbara, CA
UNIV OF CALIFORNIA / Mech Engrg Dept (Leckie), Santa Barbara, CA
UNIV OF CALIFORNIA / Mech Engrg Dept (McMeeking), Santa Barbara, CA
UNIV OF CALIFORNIA / Mech Engrg Dept (Mitchell), Santa Barbara, CA
UNIV OF CALIFORNIA / Mech Engrg Dept (Tulin), Santa Barbara, CA
UNIV OF COLORADO / CE Dept (Hon-Yim Ko), Boulder, CO
UNIV OF COLORADO / Mech Engrg Dept (Fellipe), Boulder, CO
UNIV OF COLORADO / Mech Engrg Dept (Park), Boulder, CO
UNIV OF ILLINOIS / CE Lab (Abrams), Urbana, IL
UNIV OF ILLINOIS / CE Lab (Pecknold), Urbana, IL
UNIV OF N CAROLINA / CE Dept (Gupta), Raleigh, NC
UNIV OF N CAROLINA / CE Dept (Tung), Raleigh, NC
UNIV OF TEXAS / CE Dept (Stokoe), Austin, TX
UNIV OF WYOMING / Civil Engrg Dept, Laramie, WY
WEBSTER, R / Brigham City, UT
WEIDLINGER ASSOC / F.S. Wong, Los Altos, CA